高速フーリエ変換ライブラリFFTSS

（高速関数変換研究グループ）

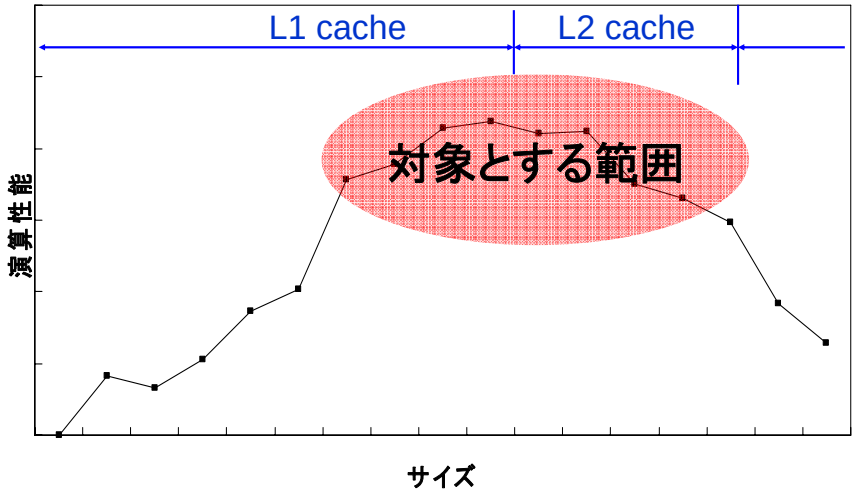
担当：額田彰（JST CREST 研究員）

はじめに

『大規模シミュレーション用基盤ソフトウェア開発』プロジェクトの高速関数変換グループの成果の一部として高速フーリエ変換ライブラリFFTSSを公開した.

FFTSSライブラリの特徴

本ライブラリはキャッシュメモリを搭載するスーパースカラ型プロセッサを対象としている. 現バージョンでは入力データ全体がキャッシュメモリにヒットするような小さなサイズで高い性能を発揮するように設計している. より大きなデータサイズ及び、2次元、3次元のFFTはこの小さなFFTと、キャッシュメモリ⇔主記憶間のコピールーチンを組み合わせて実現する.



主要なFFTライブラリー覧

現在, 非常に多種多様なFFTライブラリが入手可能である. FFTSSライブラリはこれらの性能を超えることを目標としている.

◆ベンダー提供のライブラリ

- **Intel Math Kernel Library (MKL)** for IA-32,IA-64,EM64T
- **Intel Integrated Performance Primitive (IPPS)** for IA-32,IA-64, EM64T
- **AMD Core Math Library (ACML)** for IA-32,x86-64
- **IBM Engineering and Scientific Subroutine Library (ESSL)** for PowerPC
- **SGI Scientific Computing Software Library (SCSL)** for IA-64,MIPS
- **Sun Performance Library** for UltraSPARC
- **Apple vDSP Library** for PowerPC G4/5

◆フリーソフトとして提供されているライブラリ

- **FFTW** http://www.fftw.org/
- **FFTE** http://www.ffte.jp/

FFTSSライブラリは以下のURLから入手できる.

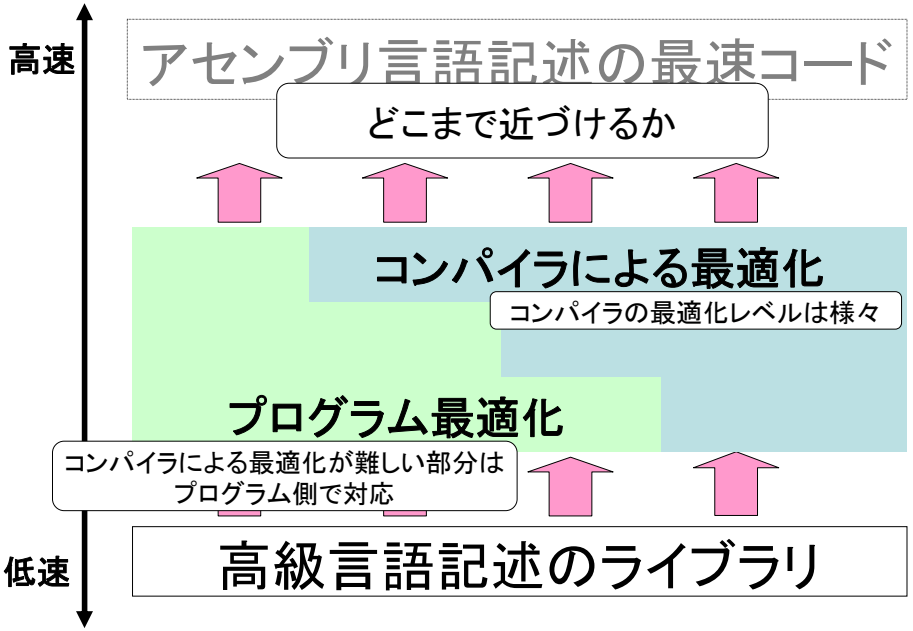
http://ssi.is.s.u-tokyo.ac.jp/fftss/

多様な計算機環境に適応するために

現在使用されている計算機のアーキテクチャは実に多種多様である．それらに適した実行コードを生成できるような条件を揃えることが重要である．

特定のプロセッサに最適化されたライブラリを作成することは比較的容易である． 最も極端な例を挙げると，アセンブリ言語で記述してしまえば良い．プロセッサの仕様を熟知していれば最速な実行コードを作成することができるだろう．しかしこれでは他のプロセッサで動作しないためポータビリティが全く無い．

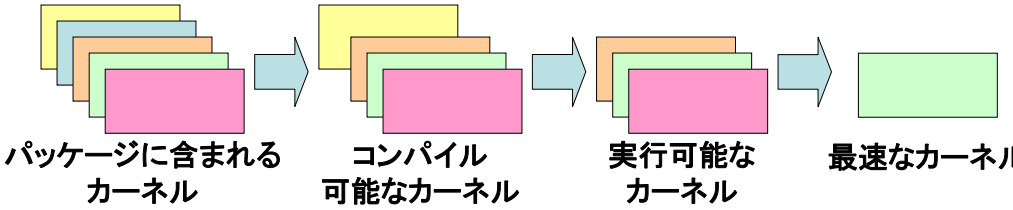
オープンソースで配布するライブラリとしては高級言語で記述し，プログラム最適化とコンパイラの最適化によって各マシンに最適に近いコードを生成することを目指す．



カーネルの選択

全てのアーキテクチャで同じFFTカーネルが適している訳ではない．例えば積和演算命令を持つプロセッサでは積和演算命令に適した別のFFTカーネルを使うべきである．このためFFTSSライブラリでは多くのFFTカーネルを搭載し，実行前のプラン作成時にその中からそのプロセッサに最適なFFTカーネルを試行によって選択する方針を採用する．

カーネル決定までの流れ



プラン構造体の最適化

プラン構造体の情報を元に計算を実行する． 特に重い処理である除算やループ以外の予測ミス率の高い条件分岐の数を削減する等の最適化を行っている．

以下は2048点FFTの実行プランの例であるが，選択されたカーネル関数への関数ポインタ及び2重ループの反復回数のリストを格納している．switch文などを使わずに関数ポインタでジャンプすることによって，多数のFFTカーネルを使用可能な状況でも速度低下を避けることができる．

2048点FFTの実行プランの例

```
{{ r8_fma_o_f, 256, 1},
 { r4_fma_o_f, 64, 8},
 { r4_fma_o_f, 16, 32},
 { r4_fma_u4_f, 4, 128},
 { r4_fma_u1_f, 1, 512}}
```

```
switch (kern) {
case FMA_O: r4_fma_o_f(); break;
case FMA_U4: r4_fma_u4_f(); break;
case FMA_U1: r4_fma_u1_f(); break;
……
}
```

コンパイラの最適化の支援

コンパイラでの自動最適化が難しい部分に関しては最適化の支援を行う.

各種拡張機能への対応

◆プロセッサ依存の拡張機能

◆FFTの計算性能に影響大　⇒　使わないと遅い

◆コンパイラが自動的に対応コードを出力しない

◆明示的な記述が必要

以下の拡張機能をサポートする.

SSE2/SSE3

◆IA-32, x86-64, EM64T 等

◆SIMD型の拡張命令. 2個の倍精度浮動小数に対して同じ演算.

積和演算

◆PowerPC, MIPS, IA-64 等

◆乗算と加減算を1命令で実行　⇒　積和演算数を少なくする

Blue Gene Double FPU

◆Blue Gene のPowerPC440D FP2専用

◆SIMOND 型命令. 2個の積和演算を1命令で実行

ひねり係数テーブル

◆Packed Load命令に対応

2個のデータを1命令でロード可能

◆独自の8基底カーネル

4基底とひねり係数テーブルを共有可能

二重ループ部分の最適化

典型的なFFTプログラム（図A）

◆反復回数は計算中に変化.

◆内側の反復回数は1になり得る

この時には極端に実行効率が低下.

従来の解決策

◆ループ交換

□内側と外側のループを交換

□ひねり係数のメモリアクセス増加

FFTではメモリアクセスの影響大.

FFTSSの手法

◆内側のループを展開

□N2が1（図B）

□N2が1以外の小さい数（図C）

図Bや図Cのようなコードになればコンパイラが十分な最適化を行うことができる.

別名参照問題

関数ポインタを多用するのでC言語

配列へのポインタの別名参照可能性

⇒コンパイラの積極的な最適化に影響

別名参照が無いことを指示する方法

①キーワード(**restrict,const**)

②コンパイラオプション(-fno-alias等)

③プラグマ指示詞(disjoint等)

多くのコンパイラで利用可能な

①キーワードによる明示を採用した.

```
COMPLEX*16 Y(N2,N1,2), X(N2,2,N1)
DO I=1,N1
   DO J=1,N2
      Y(J,I,1) = X(J,1,I) + X(J,2,I) * W(1,I)
      Y(J,I,2) = X(J,1,I) - X(J,2,I) * W(1,I)
   END DO
END DO
```

図A　二重ループのコード

```
COMPLEX*16 Y(N1,2), X(2,N1)
DO I=1,N1
   Y(I,1) = X(1,I) + X(2,I) * W(1,I)
   Y(I,2) = X(1,I) - X(2,I) * W(1,I)
END DO
```

図B　N2＝1の場合

```
COMPLEX*16 Y(2,N1,2), X(2,2,N1)
DO I=1,N1
   Y(1,I,1) = X(1,1,I) + X(1,2,I) * W(1,I)
   Y(1,I,2) = X(1,1,I) - X(1,2,I) * W(1,I)
   Y(2,I,1) = X(2,1,I) + X(2,2,I) * W(1,I)
   Y(2,I,2) = X(2,1,I) - X(2,2,I) * W(1,I)
END DO
```

図C　N2＝2の場合